

Program Synthesis from Natural Language

¹Ms.SHRUTHI M T, ² DEVIKA R N

¹Assistant Professor, Department of MCA, BIET, Davenagere

²Student, Department of MCA, BIET, Davanagere

ABSTRACT

These modern software development trends, which prioritize flexibility and speed, do work well with some programming languages. But using problem statements to write code is a function that only requires knowledge of logical ideas, not programming jargon. Conventional software development processes are more expensive and time-consuming because they use manual programming techniques to implement common best practices. An intelligent synthesis system that can automatically produce executable source code in response to natural language commands related to many computer programming languages is made possible by this project. Furthermore, the development of software requirements has been translated into common language and is not constrained by grammar or language. The current trends in software development, which prioritize speed and variety, support certain programming languages. However, using problem statements to implement code is a function that simply requires an understanding of logical principles and programming terminology. Common procedures in the traditional software development process are carried out using manual programming techniques, which increases their expense and duration. Significantly, the project enables the use of an intelligent synthesis system that can independently generate executable source code in response to commands pertaining to several computer programming languages. It is possible to create software requirements that are converted into everyday English without any restrictions on syntax or language.

I INTRODUCTION

The system is designed to bridge the gap between human thought and machine-understandable code. By allowing users to describe algorithms, operations, or full application structures in plain English, the system automatically generates code in the user's chosen programming language. It acts as an intelligent mediator between human cognitive expression and rigid programming syntax. The platform utilizes advanced NLP pipelines to extract intent, detect entities, identify control structures, and deduce data flow from text descriptions. The internal semantic engine

this extracted meaning into an abstract syntax tree (AST), which is then compiled into executable code. define variable types, classes, or logic flows manually. For example, the input Build a c roll number, and grade, and add a function to display details immediately produces implementations in Python, Java, C++, etc.1.2 Important Features 1.Natural Language Understanding Engine Breaks down difficult instructions, points out coding elements (loop, if/else, function), and pulls out algorithmic needs. 2. Multi-Language Code Generator Generates correct programs for Python, Java, C++, C, or JavaScript based on its understanding of programming models and structures.

II LITERATURE SURVEY

2.1 Existing System The traditional development method is such that the users have to write codes by hand, adhering to certain rules. The current systems fall into the following categories: 1. IDEs and code editors: Although VS Code, PyCharm, and Eclipse environments aid in the development process, programming knowledge is also essential. 2. Generators of templates: Certain systems are capable of automatically producing boilerplate code, but they are unable to comprehend the semantics of natural language. 3. NL to Code Systems Based on Rules: Only a few specific command patterns, such SQL statements, have been converted using this purely grammatical classical method. 4. Search-Based Support: This covers platforms such as GitHub Copilot and Stack Overflow. . These toolkits are helpful, but they are predicated on programming knowledge. A comprehensive natural language → executable program with semantic understanding and multilingual output is not now provided by any of these systems:

Language Dependency: Current tools specialize in one programming language rather than generating across many. 1. No End-to-End Automation Users still need to manually write logic or correct syntax errors. 2. No Real-Time Refinement Systems cannot interact conversationally to refine programs. 3. Error-Prone Code Autocompletion LLM-powered code suggestions sometimes produce hallucinated libraries or inconsistent code. 4. Limited to Professional Programmers Beginners struggle due to required prior programming knowledge.

2.2 Proposed System The proposed NL-to-Code Program Synthesis System uses modern AI and deep learning to generate full programs from natural language descriptions. It combines:

- Transformer-based encoder-decoder models
- Domain-specific datasets
- Semantic parsing
- Code verification engines

The system extracts intent, identifies required logic, and generates optimized implementations across multiple languages. It also provides interactive feedback, corrections, and documentation.

2.3 Feasibility Study Technical Feasibility • LLMs like GPT-based architectures enable accurate code generation.

- Plenty of open-source programming datasets exist.
- Cloud GPU environments support scalable model inference. Economic Feasibility
- Low operational cost once deployed.
- Possible revenue models:
 - o API subscription
 - o Enterprise licensing
 - o Education-sector integrationsOperational Feasibility
- Minimal training required for users.
- Natural language makes system highly intuitive. Legal Feasibility
- Compliance with licensing for dataset usage.
- Data privacy ensured through encryption and anonymization

2.4. Hardware & Software Requirements

Hardware • Server: 8-core CPU, 32 GB RAM, 1 TB SSD • Client: Dual-core CPU, 4 GB RAM • GPU (optional for ML training): NVIDIA RTX series • Backup: Cloud or NAS storage **Software** • OS: Windows/Linux • Languages: TypeScript Libraries: PyTorch, TensorFlow, NumPy, Transformers • Frontend: HTML/React Js • Database: MongoDB / SQLite • IDE: VS Code, PyCharm

2.6 Tools & Technologies Frontend React

Used to build the user interface for entering natural language requirements and displaying generated code. TypeScript Provides strong typing, improves code reliability, and helps catch errors during development. Backend Node.js Handles request processing, applies rules and templates, and manages communication between frontend and code processing services. Code Processing: Gemini API Interprets textual requirements and produces structured source code based on defined rules and constraints. Supporting Tools Code Templates – Predefined structures for UI components, APIs, and modules Validation Rules – Ensure syntax correctness and type safety Formatting Tools – Maintain readable and standardized output.

III.SOFTWARE DESIGN

3.1 SYSTEM PERSPECTIVE The Program Synthesis System functions as an intelligent middleware between the user and a complex backend composed of AI models, semantic engines, validators, and execution sandboxes. It sits in a multilayer architecture consisting of: 1. User Interaction Layer The layer where users submit natural language descriptions, request code generation, refine outputs, or execute generated code.

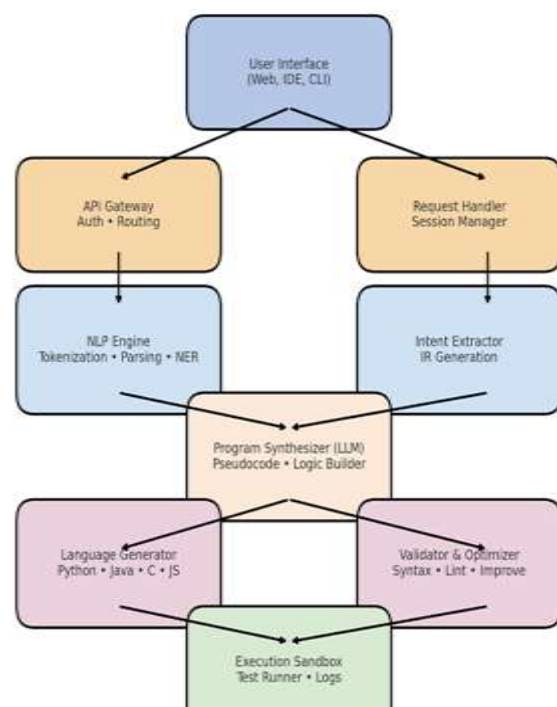
The system operates as a closed-loop architecture, where generated code can be:

1. validated, 2. executed, 3. optimized,
4. refined based on user feedback.

3.2 SYSTEM ARCHITECTURE The architecture is based on a microservice-oriented model ensuring modularity, scalability, and separation of concerns. Each module performs a well-defined role. Semantic Intent Extraction Module Translates user input into a structured Intermediate Representation (IR) language syntax

- Handles multi-step instructions
- Uses deep learning to infer logic from natural language

ARCHITECTURE DIAGRAM



IV.RESULTS

4.1 INPUT



Figure 4.1.1 : Describe the program

4.2 OUTPUT PANEL



Figure 4.2.2: Generated Output

CONCLUSION

Accurate Natural Language Understanding The system can interpret both simple and complex natural language inputs with high accuracy and extract meaningful intent from user queries.**Multi-Language Code Generation** It consistently generates executable code in several programming languages, maintaining logical consistency across languages.

Syntax and Logic Validation Built-in validation ensures that generated code is free from syntactic errors and meets expected execution outcomes.**InteractiveRefinement** Users can refine, modify, or re-generate code using natural language instructions, improving the flexibility and usability.

Secure Execution Environment By using sandboxing safely executes user-generated code without compromising system security.

Educational Utility The system line-by-line explanations of generated code, aiding students and beginners.**Scalable Architecture** A microservice-based backend employs a modular architecture, easily scalable to meet demand with more users, and is set for feature development in the future

V. FUTURE ENHANCEMENT

Although the current system already does a very good job in generating accurate and executable, there are plenty of ways to extend its functionality, efficiency, and maintainability. By working together with AI and program synthesis, this can now become more intelligent, versatile, and domain adaptive. Some of these enhancements have been outlined in this chapter for future versions of the system. 10.1 advance natural language understanding The present module of NLP is effective while handling general language inputs; however, it may not be that effective while dealing with ambiguous or abstract instructions. Improvements include the following: 1. Context-Aware Understanding Implement models that can hold conversational context to understand the multi-turn queries better.

REFERENCE

1. Sommerville, I. (2016). Software Engineering (10th ed.). Pearson Education. — Standard reference for software testing, validation, and verification.
2. Myers, G. J., Sandler, C., & Badgett, T. (2011). The Art of Software Testing (3rd ed.). Wiley. — Core reference for testing strategies and quality assurance.
3. Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating Large Language Models Trained on Code. ArXiv preprint arXiv:2107.03374. — Relevant for AI-based code generation and evaluation.
4. Yin, P., & Neubig, G. (2017). A Syntactic Neural Model for General-Purpose Code Generation. Proceedings of ACL. — Focuses on NL-to-code synthesis models.
5. Ahmad, W. U., Chakraborty, S., Ray, B., & Chang, K. (2021). Unified Pre-training for Program Understanding and Generation. NAACL-HLT. — Useful for AI-driven program synthesis systems.
6. IEEE Computer Society. (2014). IEEE Standard for Software Testing (IEEE 829 / ISO/IEC/IEEE 29119). — Official testing documentation standard.
7. OpenAI. (2023). Code Generation and Function — Reference for modern AI-based code generation systems.
8. Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A Practitioner's Approach. McGraw-Hill. — Covers testing in intelligent and complex syst